IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510103

SmartCompress AI: An Entropy-Aware Hybrid Machine Learning Framework for Intelligent Lossless Compression Algorithm Selection

Sanjiv R[1], Sriramji P[2], P. Durgadevi[3]

Department of Computer Science and Engineering,

SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[1-3]

Abstract: Organisations store ever larger volumes of logs, records, source code and backups, yet the choice of compression algorithm for each file is usually left to habit, with one archiver applied to everything. This paper presents SmartCompress AI, a web-based storage optimisation framework that inspects a file before compressing it and recommends the most suitable lossless engine for the user's goal. The system measures Shannon entropy over a streaming 64 KB window, extracts file size, category and extension, and passes these features together with a user preference (fastest, balanced or maximum compression) to a Random Forest classifier. A rule-based engine guards the classifier for pre-compressed and near-random data and acts as a fallback whenever the model is unavailable. Six engines are supported: ZIP, GZIP, BZIP2, LZMA, 7Z and PDF stream re-deflation. On a 17-file, 4.01 MB benchmark corpus, LZMA achieved the highest aggregate saving of 92.28% but needed 567.6 ms in total, about 14 times longer than ZIP (40.8 ms, 90.19% saving). The classifier reached 61.5% held-out accuracy, yet its wrong choices cost only 0.13 percentage points of saving on average, because the candidate engines are close for most compressible data. Every operation is recorded in a SQLite ledger and can be exported as a PDF audit report, and all 34 automated tests pass. The framework demonstrates that a lightweight, explainable recommender can make compression choices transparent and resource-aware, supporting sustainable data storage.

Keywords: Lossless Compression, Shannon Entropy, Algorithm Selection, Random Forest, Hybrid Recommender, LZMA, DEFLATE, Storage Optimisation, Flask, Sustainable Computing

I. INTRODUCTION

A. Background and Motivation

Global data volumes continue to grow faster than storage budgets. IDC projected the world's data to exceed one hundred zettabytes in the current decade [1], and every stored byte consumes disk capacity, backup bandwidth and, ultimately, electricity [2], [5]. Lossless compression is the cheapest tool for reducing this footprint, and mature algorithms such as DEFLATE [8], [9], the Burrows-Wheeler transform [10] and LZMA [11] are freely available on every modern platform. The difficulty is not the absence of algorithms but the absence of guidance in choosing between them. Each engine trades ratio for speed differently, and the trade-off depends on the data. Highly repetitive logs shrink by more than 99% under any engine, whereas encrypted payloads and existing archives cannot be compressed at all and only waste CPU time. Users rarely know which situation they are in, so they apply one default archiver everywhere and either leave savings unrealised or spend processing time for nothing.

SmartCompress AI addresses this gap. It measures the information content of a file with Shannon entropy [3], combines it with simple metadata and the user's stated priority, and recommends an engine using a Random Forest classifier [15] protected by explicit rules. The recommendation is explained in plain language, the compression is performed, and the result is logged for later audit.

B. Research Contributions

This paper makes four contributions to practical storage optimisation:

(i) We design an entropy-aware feature set (file size, Shannon entropy, category, extension and user preference) that is computed in a single streaming pass and is sufficient to drive engine selection.

(ii) We propose a hybrid recommender in which a Random Forest classifier is guarded by a rule-based engine that handles pre-compressed and near-random data and provides a fallback when the model is unavailable.

(iii) We implement the complete pipeline as a Flask web application with six compression engines, batch processing, a SQLite history ledger and PDF audit reports.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510103

(iv) We benchmark all engines on a mixed 17-file corpus and evaluate the recommender in terms of both classification accuracy and the practical cost of its errors.

II. LITERATURE REVIEW

Modern lossless compression rests on the information theory of Shannon [3], who showed that the entropy of a source bounds the average number of bits needed to encode it. Huffman coding [6] attains this bound for symbol-wise codes, while the dictionary methods of Ziv and Lempel [7] exploit repetition across longer strings. DEFLATE combines LZ77 matching with Huffman coding and underlies both ZIP and GZIP [8], [9]. Burrows and Wheeler [10] introduced a block-sorting transform that groups similar contexts together, which BZIP2 uses to obtain higher ratios on text at the cost of speed. LZMA [11] extends the LZ family with a large sliding dictionary and range coding and typically delivers the best ratios among general-purpose compressors, which is why it powers the 7Z format. Surveys by Sayood [12] and Deorowicz [22] document the ratio-speed trade-offs among these families, and newer designs such as Zstandard [21] offer tunable points along the same curve.

Adapting compression to the data has been explored mainly in systems research. Krintz and Sucu [13] proposed adaptive on-the-fly compression that predicts whether compressing network data will reduce total transfer time. Harnik et al. [14] showed that a quick estimate of compressibility avoids wasting CPU on incompressible blocks in real-time storage systems. These works confirm that a cheap pre-analysis step pays off, but they decide only whether to compress, not which algorithm to use, and they are embedded in storage controllers rather than exposed to end users.

On the learning side, Random Forests [15] suit small tabular datasets with mixed numeric and categorical features, provide feature-importance estimates and resist overfitting better than single decision trees, and mature implementations exist in scikit-learn [16]. To the best of our knowledge, no openly available end-user tool combines entropy-based pre-analysis, a learned engine selector and a transparent audit trail. SmartCompress AI is designed to fill this gap.

III. METHODOLOGY

A. System Architecture Overview

SmartCompress AI is organised in four layers, shown in Fig. 1. The presentation layer is a Flask [17] web interface with single-file upload, a Batch Studio for folders, a history dashboard and progressive-web-app support. The service layer validates uploads, assigns UUID-based storage names and coordinates analysis, recommendation, compression and reporting. The engine layer contains the analyser, the hybrid recommender and the compression engines built on the Python standard library [18], with PyMuPDF [19] for PDF stream re-deflation. The persistence layer stores every operation in a SQLite *compression_history* table and produces PDF audit reports with ReportLab [20].

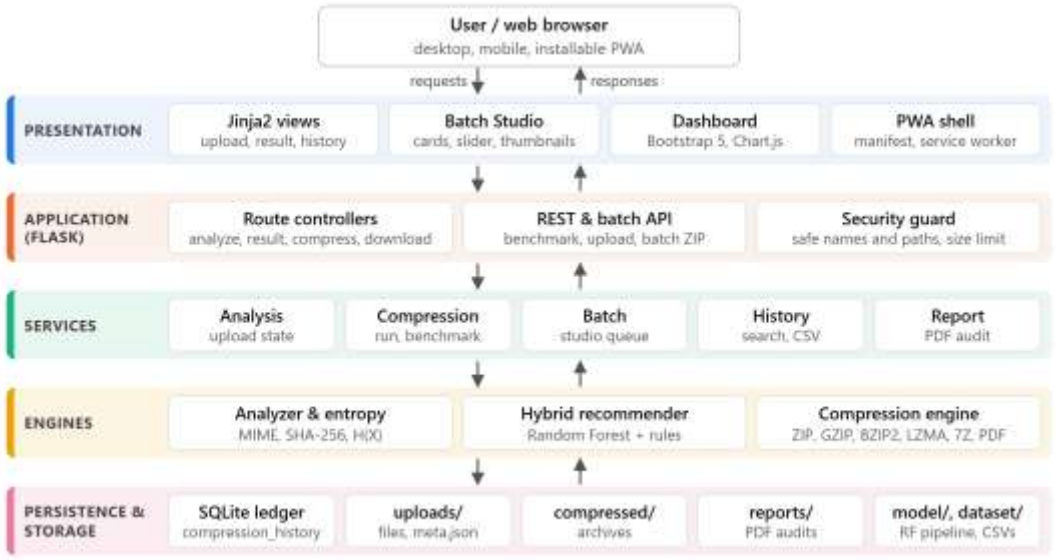


Fig. 1. System architecture of SmartCompress AI

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510103

B. Entropy-Based File Analysis

For each uploaded file the analyser reads the content in 64 KB chunks, so memory use stays constant regardless of file size. It accumulates a 256-bin byte histogram and computes the Shannon entropy

$$H = -\Sigma_{i=0\ldots255}\, p_i \log_2 p_i \quad (1)$$

where p_i is the relative frequency of byte value i. The result lies between 0 bits/byte (a single repeated byte) and 8 bits/byte (uniformly random data). In the same pass the analyser records the size, extension, MIME type and SHA-256 digest, and maps the extension to a category such as *Text / Document*, *Structured Data / Code* or *Binary / Executable*. Files at or above 7.92 bits/byte, and recognised archive formats, are flagged as pre-compressed.

C. Hybrid Recommendation Engine

The recommender receives five features: file size, entropy, category, extension and preference, with the categorical features label-encoded. A Random Forest classifier predicts one of five engines (ZIP, GZIP, BZIP2, LZMA, 7Z) and reports the share of trees voting for the winner as its confidence. The rule-based engine runs on every request and serves three purposes. First, for pre-compressed files at the entropy ceiling it overrides the model and returns a safety notice recommending fast GZIP packaging, because no engine can reduce such data. Second, if the model file is missing or inference fails, its recommendation is used directly. Third, its output is shown alongside the model's so that the user can compare the two. PDF inputs are routed to the dedicated PDF engine, which re-deflates content streams while keeping the document valid.

Training labels were generated empirically. Each of the 17 corpus files was compressed with every engine under each of the three preferences, giving 51 records, and the label was the engine that best served the stated preference: the smallest output for *maximum compression*, the shortest time for *fastest*, and the best combined score for *balanced*. Table I lists the features and their learned importances.

TABLE I RECOMMENDER FEATURES AND RANDOM FOREST IMPORTANCE

Feature	Type	Source	Importance
Extension	Categorical	File name	26.7%
Preference	Categorical	User choice	22.8%
Shannon entropy	Numeric (0-8)	Byte histogram	20.6%
File size	Numeric (bytes)	File system	17.3%
Category	Categorical	Extension map	12.6%

D. Compression, Ledger and Reporting

After the user accepts or overrides the recommendation, the selected engine writes the output to a dedicated folder, and the service records original and compressed sizes, space saving, time taken, the recommendation source and the confidence in the ledger. Space saving is computed as (1 − compressed size / original size) × 100%. The dashboard aggregates these records, and any operation can be exported as a PDF audit report that states the entropy, the reasoning behind the recommendation and the measured result.

IV. RESULTS AND EVALUATION

A. Experimental Setup

The benchmark corpus contains 17 files totalling 4,012,869 bytes, chosen to span the full entropy range: server and trace logs, CSV telemetry and a financial ledger, JSON records, an SQL dump, Python, JavaScript, HTML and Markdown sources, sparse and firmware binaries, a compiled executable, an encrypted payload and an existing ZIP archive. Entropy ranges from 0.15 bits/byte (sparse memory image) to 8.00 bits/byte (encrypted and archived data). Every file was compressed with each engine at its default level, and timings are wall-clock times measured inside the application. All experiments ran on a desktop with an AMD Ryzen 5 5600X processor and 32 GB of RAM under Windows 11 Pro with Python 3.14. No GPU was used.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510103

B. Compression Engine Performance

Table II summarises the results per engine across the corpus. The LZMA-based engines achieve the best aggregate saving, while the DEFLATE-based engines are roughly 14 times faster for a loss of only about two percentage points.

TABLE II ENGINE PERFORMANCE ON THE 17-FILE BENCHMARK CORPUS

Engine	Aggregate Saving	Output Size	Total Time	Best-Ratio Wins	Fastest Wins
ZIP	90.19%	384.3 KB	40.8 ms	0	12
GZIP	90.23%	382.7 KB	39.1 ms	3	5
BZIP2	91.67%	326.3 KB	343.3 ms	3	0
LZMA	92.28%	302.6 KB	567.6 ms	9	0
7Z	92.26%	303.3 KB	558.2 ms	2	0

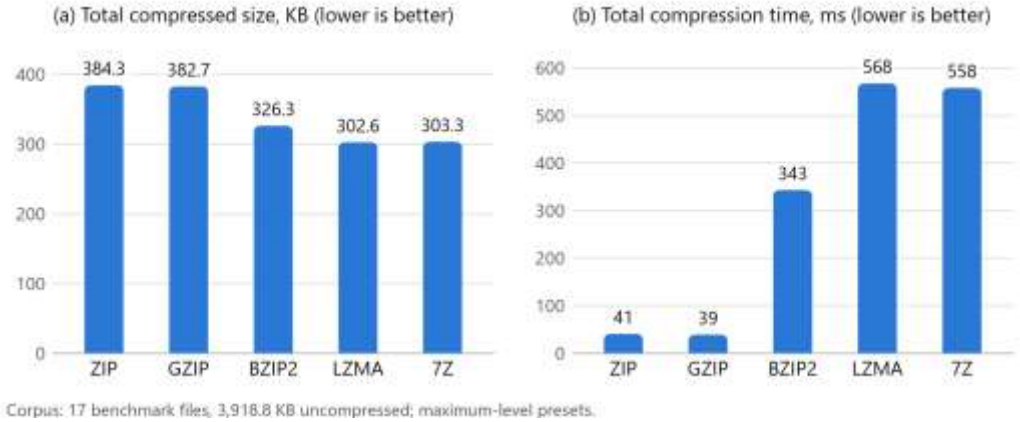


Fig. 2. Total compressed size and compression time per engine on the benchmark corpus

Fig. 2 shows the ratio-speed trade-off directly, and Fig. 3 relates entropy to the best achievable saving. Files below about 5 bits/byte compress by more than 96% with every engine, so the choice of engine there mainly affects speed. Above 7.5 bits/byte, savings collapse: the compiled executable saved only 11.77%, and the encrypted payload and existing archive grew slightly (−0.05% and −0.08%). This boundary is exactly where the rule-based guard steps in.

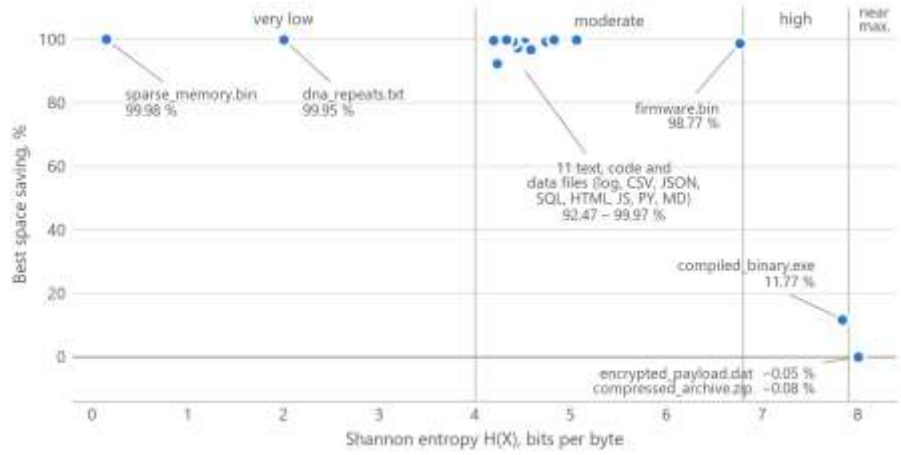


Fig. 3. Best space saving versus Shannon entropy for the benchmark files

C. Recommender Accuracy

Table III reports the classifier metrics. On the 13-record held-out split the Random Forest predicted the empirically best engine in 61.5% of cases, while 4-fold cross-validation gave 42.9% ± 7.8%. The gap shows that the 51-record dataset is

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510103

too small for a stable estimate and that the model is partly overfitted. Accuracy alone, however, overstates the cost of a mistake. Across the test predictions, using the predicted engine instead of the best one reduced saving by only 0.13 percentage points on average, because for compressible files several engines finish within a fraction of a percent of each other.

TABLE III RANDOM FOREST RECOMMENDER EVALUATION

Metric	Value	Notes
Training records	51	17 files × 3 preferences
Held-out test accuracy	61.54%	13 records
4-fold cross-validation	42.9% ± 7.8%	Indicates overfitting
Mean saving gap on errors	0.13 pp	Predicted vs. best engine
Automated tests	34 / 34 passed	pytest, 2.27 s

D. Failure Mode Analysis

Three failure modes were observed. First, under the *fastest* preference ZIP and GZIP differ by fractions of a millisecond, so timing noise alone can change the label, and the model frequently confuses the two. Second, the classifier relies heavily on file extension, so a renamed or unusual extension can push it toward a poor choice; entropy only partly compensates. Third, for high-entropy binaries just below the ceiling, such as the compiled executable, the model occasionally suggests LZMA, which costs about ten times more time than GZIP for a similar saving. A larger training set and a cost-aware objective would address all three.

V. SYSTEM IMPLEMENTATION AND DEPLOYMENT

A. Flask Application Architecture

The application is built with Flask and organised into routes, services, engines and utilities. Uploads are checked against an extension whitelist and a configurable size limit and stored under UUID-based names to prevent path traversal and filename collisions. The Batch Studio accepts whole folders, analyses each file and compresses them either individually or into a single archive. A cleanup utility removes expired uploads and outputs, and a progressive-web-app manifest allows the interface to be installed on desktop and mobile browsers.

B. Response Structure and Audit Trail

Each analysis returns a JSON object containing the file metadata, the entropy and its interpretation, the recommended engine, the recommendation source (Random Forest or rule-based), the model confidence with per-engine probabilities, and a plain-language explanation. The *compression_history* ledger stores these fields with the measured outcome, so the dashboard can report total bytes saved, engine usage and average saving, and every result can be traced through its audit report.

VI. DISCUSSION

The results support the central idea of SmartCompress AI: a few cheap features are enough to make compression decisions sensible and explainable. Entropy cleanly separates data that will compress well from data that will not, which lets the system avoid wasted work on encrypted and archived files. For compressible data, the user's preference matters more than the precise choice of engine, which is why the classifier's modest accuracy translates into a negligible loss of saving.

The hybrid design is a practical strength. The rules encode knowledge that holds regardless of the training data, such as the futility of compressing random bytes, while the model learns the finer trade-offs among engines. Because both outputs are shown with confidence values, users can see why a recommendation was made and override it when they know better. The work also aligns with the UN Sustainable Development Goals [4]: smaller archives reduce storage hardware and energy use (SDG 12 and SDG 13), and an open, low-cost tool supports resilient digital infrastructure (SDG 9).

The main limitation is the size of the training set. With 51 records the cross-validated accuracy is low and unstable, and the corpus does not cover multimedia or very large files. Timing labels are also sensitive to machine load. Future work

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026
DOI: 10.17148/IJARCCE.2026.1510103

will enlarge the dataset, add Zstandard [21] and Brotli engines with tunable levels, train with a cost-sensitive objective that weighs saving against time, and sample entropy in parallel for multi-gigabyte inputs.

VII. CONCLUSION

This paper presented SmartCompress AI, an entropy-aware framework that recommends and applies the most suitable lossless compression engine for each file. Shannon entropy and simple metadata, combined with the user's preference, drive a Random Forest classifier guarded by explicit rules for pre-compressed and near-random data. On a 4.01 MB mixed corpus, LZMA delivered the highest saving of 92.28%, while the DEFLATE-based engines were about 14 times faster at roughly 90% saving, confirming that no single engine is best for every goal. The recommender reached 61.5% held-out accuracy, and its errors cost only 0.13 percentage points of saving on average.

Implemented as a Flask web application with batch processing, a SQLite ledger and PDF audit reports, and verified by 34 passing tests, the system makes compression choices transparent and resource-aware. The identified failure modes point to clear extensions: a larger and more diverse training set, cost-sensitive learning and additional modern engines. With these, SmartCompress AI can become a practical everyday tool for sustainable storage management.

REFERENCES

[1]. D. Reinsel, J. Gantz, and J. Rydning, *The digitization of the world: From edge to core*, IDC White Paper US44413318, Nov. 2018.
[2]. EMC Education Services, *Information Storage and Management*, 2nd ed., Wiley, 2012.
[3]. C. E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal, vol. 27, no. 3, pp. 379-423, 1948.
[4]. United Nations General Assembly, *Transforming our world: The 2030 Agenda for Sustainable Development*, A/RES/70/1, 2015.
[5]. International Energy Agency, *Electricity 2024: Analysis and forecast to 2026*, IEA, Paris, 2024.
[6]. D. A. Huffman, *A method for the construction of minimum-redundancy codes*, Proc. IRE, vol. 40, no. 9, pp. 1098-1101, 1952.
[7]. J. Ziv and A. Lempel, *A universal algorithm for sequential data compression*, IEEE Trans. Information Theory, vol. 23, no. 3, pp. 337-343, 1977.
[8]. P. Deutsch, *DEFLATE compressed data format specification version 1.3*, RFC 1951, IETF, 1996.
[9]. P. Deutsch, *GZIP file format specification version 4.3*, RFC 1952, IETF, 1996.
[10]. M. Burrows and D. J. Wheeler, *A block-sorting lossless data compression algorithm*, Digital Equipment Corp. SRC Research Report 124, 1994.
[11]. I. Pavlov, *LZMA SDK (Software Development Kit)*, 7-Zip. [Online]. Available: https://www.7-zip.org/sdk.html
[12]. K. Sayood, *Introduction to Data Compression*, 5th ed., Morgan Kaufmann, 2017.
[13]. C. Krintz and S. Sucu, *Adaptive on-the-fly compression*, IEEE Trans. Parallel and Distributed Systems, vol. 17, no. 1, pp. 15-24, 2006.
[14]. D. Harnik, R. Kat, O. Margalit, D. Sotnikov, and A. Traeger, *To zip or not to zip: Effective resource usage for real-time compression*, Proc. USENIX FAST, pp. 229-241, 2013.
[15]. L. Breiman, *Random forests*, Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.
[16]. F. Pedregosa et al., *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.
[17]. Pallets Projects, *Flask documentation (3.x)*. [Online]. Available: https://flask.palletsprojects.com/
[18]. Python Software Foundation, *Data compression and archiving*, Python 3 documentation. [Online]. Available: https://docs.python.org/3/library/archiving.html
[19]. Artifex Software, *PyMuPDF documentation*. [Online]. Available: https://pymupdf.readthedocs.io/
[20]. ReportLab, *ReportLab PDF library user guide*. [Online]. Available: https://docs.reportlab.com/
[21]. Y. Collet and M. Kucherawy, *Zstandard compression and the application/zstd media type*, RFC 8878, IETF, 2021.
[22]. S. Deorowicz, *Universal lossless data compression algorithms*, Ph.D. dissertation, Silesian University of Technology, 2003.

 This work is licensed under a Creative Commons Attribution 4.0 International License